09982786h60 "0207020

FIG. 1

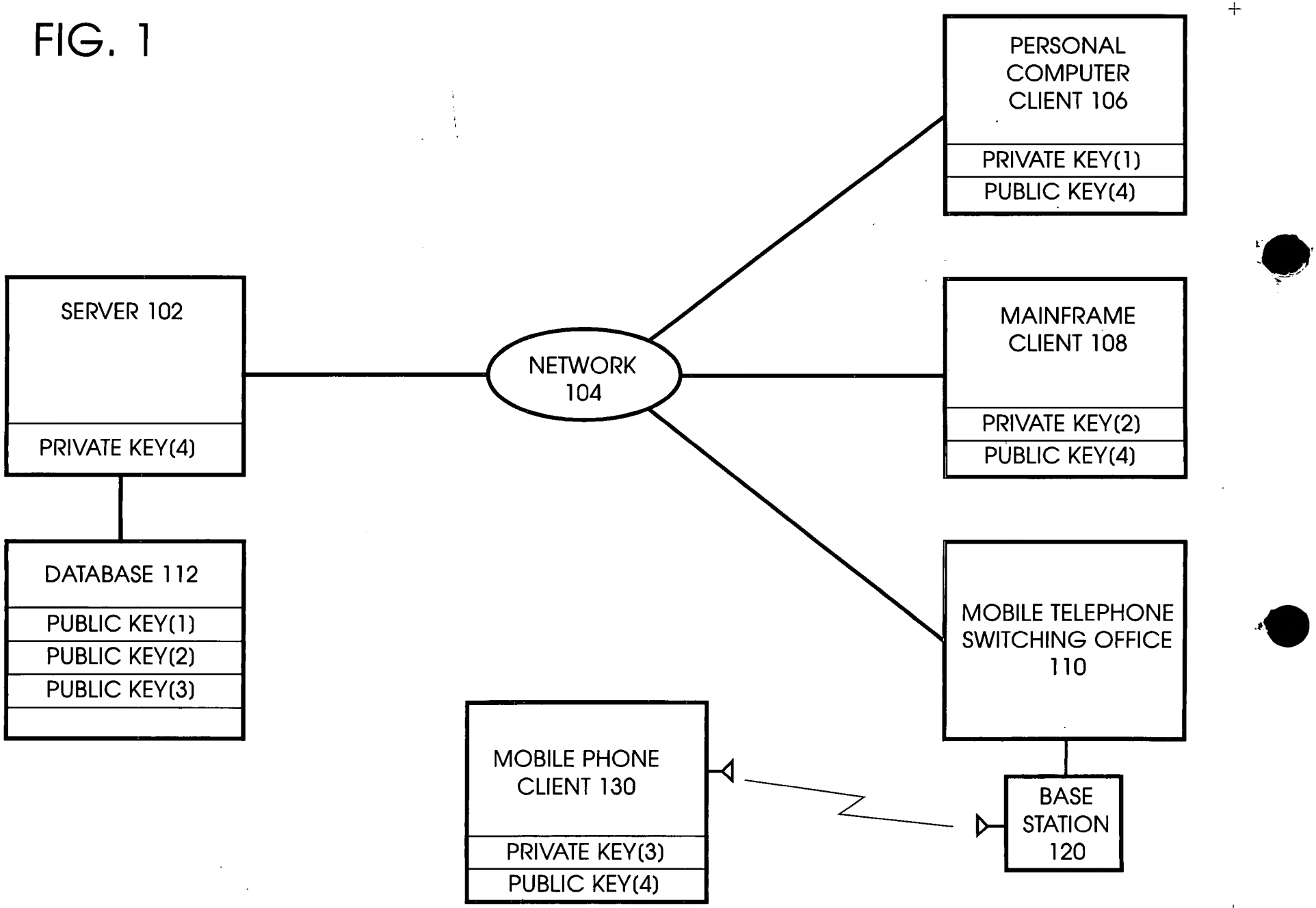


FIG. 2

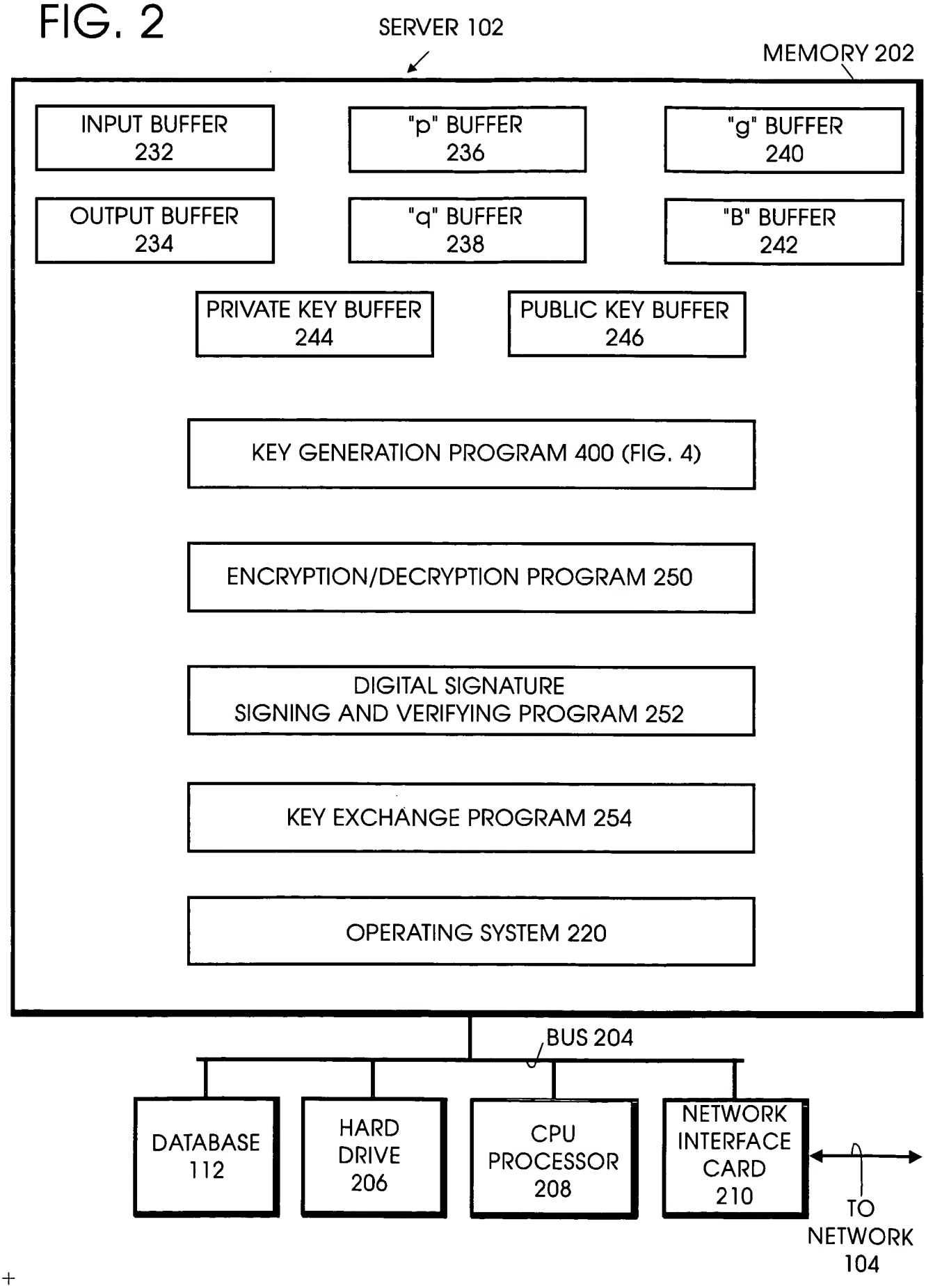
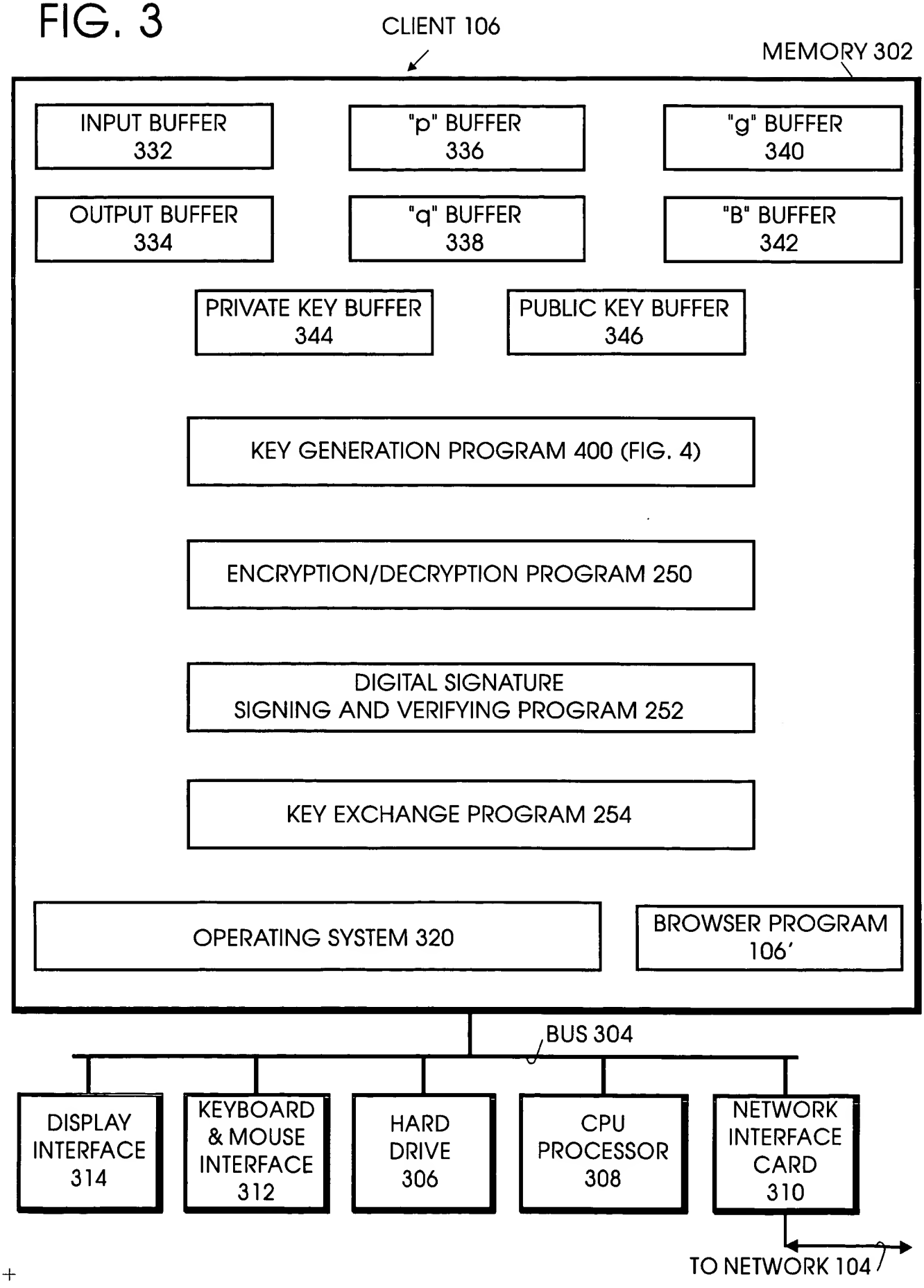
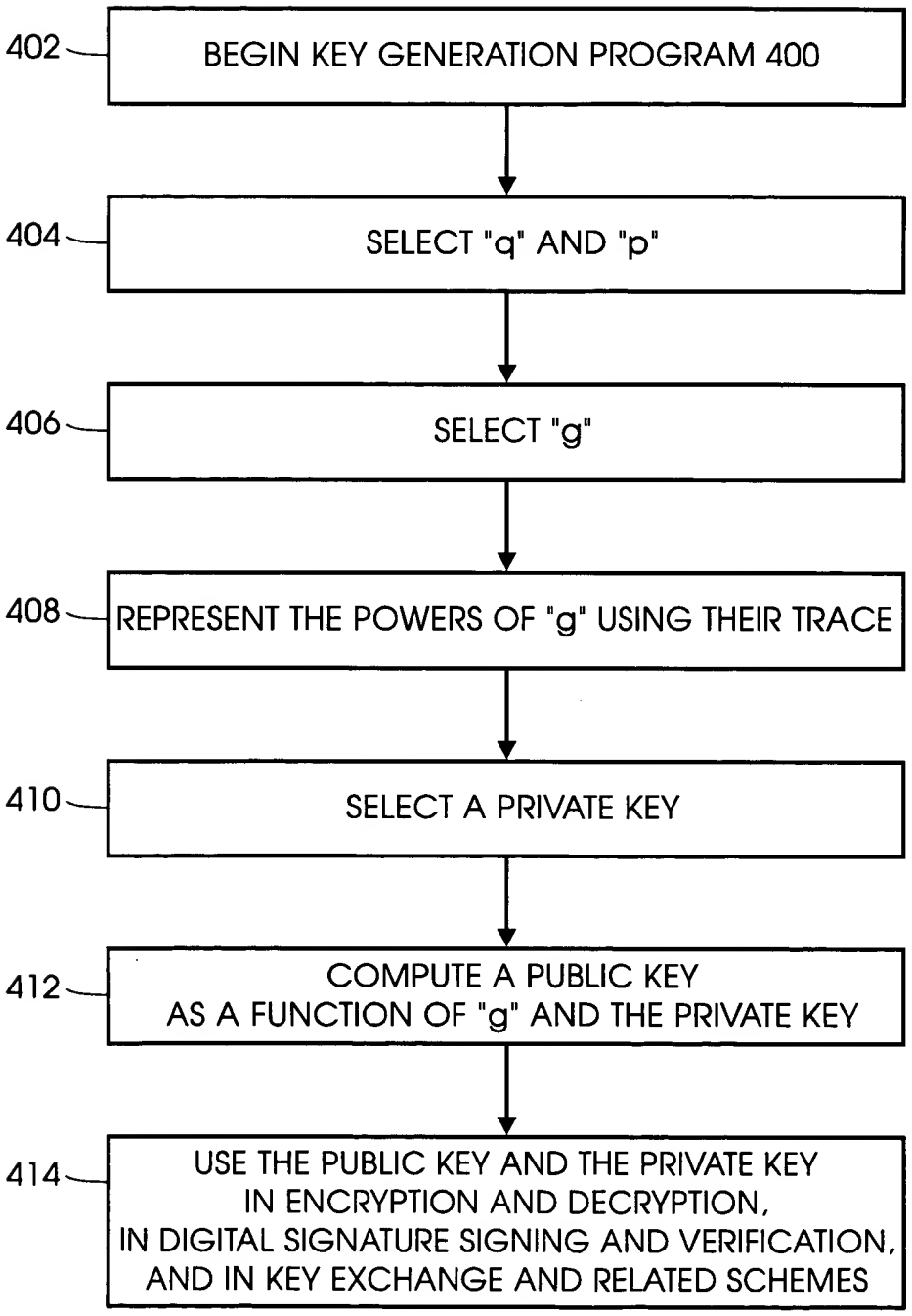

SERVER 102
MEMORY 202
INPUT BUFFER 232
"p" BUFFER 236
"g" BUFFER 240
OUTPUT BUFFER 234
"q" BUFFER 238
"B" BUFFER 242
PRIVATE KEY BUFFER 244
PUBLIC KEY BUFFER 246
KEY GENERATION PROGRAM 400 (FIG. 4)
ENCRYPTION/DECRYPTION PROGRAM 250
DIGITAL SIGNATURE SIGNING AND VERIFYING PROGRAM 252
KEY EXCHANGE PROGRAM 254
OPERATING SYSTEM 220
BUS 204
DATABASE 112
HARD DRIVE 206
CPU PROCESSOR 208
NETWORK INTERFACE CARD 210
TO NETWORK 104

FIG. 3



09420020 - 051898

FIG. 4



DOCKET NO. 9-4-9

Let $p \equiv 2 \bmod 3$ be a prime number such that $6*\log_2(p) \approx 1024$ and such that $\phi_6(p) = p^2 - p + 1$ has a prime factor q with $\log_2(q) \geq 160$. Such p and q (or of any other reasonable desired size) can quickly be found by picking a prime $q \equiv 7 \bmod 12$, by finding the two roots r_1 and r_2 of $x^2 - x + 1 \equiv 0 \bmod q$, and by finding an integer k such that $r_i + k*q$ is 2 mod 3 and prime for $i = 1$ or 2. If desired, primes q can be selected until the smallest or the largest root is prime, or any other straightforward variant that fits one's needs may be used, for instance to get $\log_2(q) \approx 180$ and $6*\log_2(p) \approx 3000$, i.e., $\log_2(p)$ considerably bigger than $\log_2(q)$. From $q \equiv 7 \bmod 12$ it follows that $q \equiv 1 \bmod 3$ so that, with quadratic reciprocity, $x^2 - x + 1 \equiv 0 \bmod q$ has two roots. It also follows that $q \equiv 3 \bmod 4$ which implies that those roots can be found using a single $((q+1)/4)^{th}$ powering modulo q.

By $g \in GF(p^6)$ we denote an element of order q. It is well known that g is not contained in any proper subfield of $GF(p^6)$ (cf. [4]). In the next section it is shown that there no need for an actual representation of g and that arithmetic on elements of $GF(p^6)$ can be entirely avoided. Thus, there is no need to represent elements of $GF(p^6)$, for instance by constructing an irreducible 3^{rd} degree polynomial over $GF(p^2)$. A representation of $GF(p^2)$ is needed however. This is done as follows.

From $p \equiv 2 \bmod 3$ it follows that $p \bmod 3$ generates $GF(3)^*$, so that the zeros α and α^p of the polynomial $(X^3 - 1)/(X - 1) = X^2 + X + 1$ form an optimal normal basis for $GF(p^2)$ over $GF(p)$. Because $\alpha^i = \alpha^{i \bmod 3}$, an element $x \in GF(p^2)$ can be represented as $x_0\alpha + x_1\alpha^p = x_0\alpha + x_1\alpha^2$ for $x_0, x_1 \in GF(p)$, so that $x^p = x_0^p\alpha^p + x_1^p\alpha^{2p} = x_1\alpha + x_0\alpha^2$.

Figure 5 is a flow diagram of the method for selection of "p", as shown in section 2.1.

Docket No. 12012

Algorithm for the computation of $T(n)$ given $B = T(1)$. Given B (and B^p), we show how $S(n+1)$ and $S(2n)$ can be computed based on $S(n)$. Computation of $T(n)$ for arbitrary n then follows using the ordinary square and multiply method based on $S(1) = (B^p, 3, B)$ (cf. Definition 2.4.3).

- $S(n+1)$ can be computed from $S(n)$ using Corollary 2.4.2.ii. This takes two multiplications in GF(p^2).

- $S(2n)$ can be computed by first using Corollary 2.4.2.i to compute $T(2n-2)$ and $T(2n)$ given $S(n)$, at the cost of two squarings in GF(p^2), followed by an application of Corollary 2.4.2.iii to compute $T(2n-1)$ at the cost of two multiplications in GF(p^2).

In both steps we use that pth powering is for free in GF(p^2).

Figure 6 is a flow diagram of the arithmetic method to support key generation, as shown in section 2.4.4.

0020-3715-0409

Algorithm 3.3.8 for the computation of B.

1. Pick at random an element $B' \in \mathrm{GF}(p^2)^* \backslash \mathrm{GF}(p)^*$;
2. Use Algorithm 2.4.7 with B replaced by B' and T replaced by V to compute $V(p+1)$ (i.e., with $B' = T(1) = V(1)$);
3. If $V(p+1) \in \mathrm{GF}(p)$, then return to Step 1;
4. Use Algorithm 2.4.7 with B replaced by B' to compute $T((p^2-p+1)/q)$ (i.e., with $B' = T(1)$);
5. If $T((p^2-p+1)/q) = 3$, then return to Step 1;
6. Let $B = T((p^2-p+1)/q)$.

Figure 7 is a flow diagram of the method of key generation, as shown in section 3.3.8.

Document: 0056376 - 02/09/2005

4.1 Application to the Diffie-Hellman scheme

Suppose that two parties, Alice and Bob, who both have access to the public key data p, q, B want to agree on a shared secret key. They can do this by performing the following variant of the Diffie-Hellman scheme:

1. Alice selects at random an integer a, $1 < a < q - 2$, uses Algorithm 2.4.7 to compute $V_A = T(a) \in \mathrm{GF}(p^2)$, and sends V_A to Bob.
2. Bob receives V_A from Alice, selects at random an integer b, $1 < b < q - 2$, uses Algorithm 2.4.7 to compute $V_B = T(b) \in \mathrm{GF}(p^2)$, and sends V_B to Alice.
3. Alice receives V_B from Bob, and uses Algorithm 2.4.8 with B replaced by V_B (i.e., with $V_B = T(1)$) to compute $K_{AB} = T(a) \in \mathrm{GF}(p^2)$.
4. Bob uses Algorithm 2.4.8 with B replaced by V_A (i.e., with $V_A = T(1)$) to compute $K_{AB} = T(b) \in \mathrm{GF}(p^2)$.

The length of the messages exchanged in this DH variant is about one third of the length of the messages in other implementations of the DH scheme that achieve the same level of security and that are based on the difficulty of computing discrete logarithms in (a subgroup of) the multiplicative group of a finite field. Also, our variant of the DH scheme requires considerable less computation than those previously published methods (cf. Remark 2.4.10).

Figure 8 is a flow diagram of the method of Diffie Hellman key exchange, as shown in section 4.1, using keys generated by the method of Figure 7.

Document: 0050469 Version: 0 Type: 0

4.2 Application to the ElGamal encryption scheme

Suppose that Alice is the owner of the public key data p, q, B, and that Alice has selected a secret integer k and computed the corresponding public value $C = T(k)$ using Algorithm 2.4.7. Thus, Alice's public key data consists of (p, q, B, C). Given Alice's public key (p, q, B, C) Bob can encrypt a message M intended for Alice using the following variant of ElGamal encryption:

1. Bob selects at random an integer b, $1 < b < q - 2$;
2. Bob uses Algorithm 2.4.7 to compute $V_B = T(b) \in GF(p^2)$;
3. Bob uses Algorithm 2.4.7 with B replaced by C (i.e., with $C = T(1)$) to compute $K = T(b) \in GF(p^2)$;
4. Bob uses K to encrypt M, resulting in the encryption E.
5. Bob sends (V_B,E) to Alice.

Note that Bob may have to hash the bits representing K down to a suitable encryption key length.

Upon receipt of (V_B,E), Alice decrypts the message in the following manner:

1. Alice uses Algorithm 2.4.7 with B replaced by V_B (i.e., with $V_B = T(1)$) to compute $K = T(k) \in GF(p^2)$;
2. Alice uses K to decrypt E resulting in M.

The message (V_B,E) sent by Bob consists of the actual encryption E, whose length strongly depends on the length of M, and the overhead V_B, whose length is independent of the length of M. The length of the overhead in this variant of the ElGamal encryption scheme is about one third of the length of the overhead in other implementations of message-length independent ElGamal encryption (cf. Remark 4.2.1). Also, our method is considerably faster (cf. Remark 2.4.10).

Figure 9 is a flow diagram of the method of ElGamal encryption, as shown in section 4.2, using keys generated by the method of Figure 7.

Document: 12012_1 Page 9 of 15

Algorithm 2.5.3 for the computation of the representation of $g^a * y^b$ for integers a, b with $1 < a, b < q$, given the representation B of g and the representations C, C_+, and C_- of y, $y*g$, and y/g, respectively.

1. Compute $c = a/b \bmod q$;
2. Given B use Algorithm 2.4.7 to compute $T(c+1)$, $T(c)$, $T(c-1)$ (note that the final applications of Corollary 2.4.2.i in Algorithm 2.4.7, if any, should be replaced by the usual calculation of the full $S(2n)$);
3. Use Lemma 2.5.2 with $T(0) = 3$, $T(1) = B$, $T(-1) = B^p$, $T(c)$, $T(c+1)$, and $T(c-1)$ to compute A^c;
4. Use Lemma 2.5.2 with $T(0) = 3$, $T(1) = B$, $T(-1) = B^p$, $T(k) = C$, $T(c+1) = C_+$, and $T(c-1) = C_-$ to compute the corresponding power of A, which we denote by A^k, even though k is unknown;
5. Compute A^{c+k};
6. Using Lemma 2.5.1 and A^{c+k} compute $T(c + k)$;
7. Use Algorithm 2.4.7 with B replaced by $T(c + k)$ and n replaced by b to compute the representation $T((c + k) * b) = T(a + k * b)$ of $g^a * y^b$.

Figure 10A is a flow diagram of the arithmetic method to support generating digital signatures, as shown in section 2.5.3.

00420297.092502

4.3 Application to digital signature schemes

Let, as in 4.2, Alice's public key data consists of (p, q, B, C), where $C = T(k)$ and k is Alice's private key. Furthermore, assume that $C_+ = T(k+1)$ and $C_- = T(k-1)$ are included in Alice's public key (cf. 2.5). We show how the Nyberg-Rueppel (NR) message recovery signature scheme can be implemented using our subgroup representation. Application of our method to other digital signature schemes goes in a similar way. To sign a message M containing an agreed upon type of redundancy, Alice does the following:

1. Alice selects at random an integer a, $1 < a < q - 2$;
2. Alice uses Algorithm 2.4.7 to compute $V_A = T(a) \in \mathrm{GF}(p^2)$;
3. Alice uses V_A to encrypt M, resulting in the encryption E.
4. Alice computes the (integer valued) hash h of E.
5. Alice computes $s = (k * h + a)$ modulo q in the range $\{0,1, \ldots, q-1\}$.
6. Alice's resulting signature on M is (E,s).

As in 4.2 Alice may have to hash the bits representing V_A down to a suitable encryption key length.

To verify Alice's signature (E,s) and to recover the signed message M, Bob does the following:

1. Bob obtains Alice public key data (p, q, B, C, C_+, C_-).
2. Bob checks that $0 \leq s < q$; if not failure.
3. Bob computes the hash h of E (using the same hash function used by Alice).
4. Bob replaces h by $-h$ modulo q (i.e., in the range $\{0,1, \ldots, q-1\}$).
5. Bob uses Algorithm 2.5.3 to compute the representation V_B of $g^s * y^h$ given $a = s$, $b = h$, B, C, C_+, and C_-.
6. Bob uses V_B to decrypt E resulting in the message M.
7. If M contains the agreed upon type of redundancy, then the signature is accepted; if not the signature is rejected.

Figure 10B is a flow diagram of the method of generating digital signatures, as shown in section 4.3., using keys generated by the method of Figure 7.

0045698 = 0020040